Memoria Técnica: Sistema de Reconocimiento de Cartas de Póker

Reto de Visión Artificial - Examen Parcial

Autor: Adrian Pérez Bahamontes
Fecha: Noviembre 2025
Asignatura: Inteligencia Artificial

---

1. Hardware

1.1 Características Técnicas

Componente	Especificación
Cámara	Smartphone Android (Poco F5) conectado mediante Camo
Resolución de captura	1280 x 720 píxeles (HD)
Tasa de frames	30 FPS
Ordenador	Windows 11
Superficie de fondo	Tapete verde

1.2 Justificación del Hardware

Cámara (Smartphone + Camo):
Elegí utilizar mi smartphone Android como cámara mediante la aplicación Camo por varias razones:

1. **Facilidad de conexión**: Fue la aplicación que más simple hace la conexión de una camara entre Android y Windows.
2. **Flexibilidad de posicionamiento**: Al no utilizar una cámara dedicada, moverla se hace sencillo.

Tapete verde:
Utilicé un tapete verde como superficie de fondo porque lo requería el propio proyecto, además de que el tapete hace un buen trabajo de separar la carta del fondo.

---

2. Software

2.1 Características Técnicas

Componente	Versión	Propósito
Python	3.12.6	Lenguaje de programación principal
OpenCV	4.12.0	Procesamiento de imágenes y visión artificial
NumPy	2.2.6	Operaciones matriciales y numéricas
Camo	Última versión	Conexión cámara smartphone-PC

2.2 Justificación del Software

Python + OpenCV + NumPy:

La decisión de usar este conjunto de lenguaje y librerías es muy obvio. Python es la columna vertebral de toda visión artificial hoy en día, y el resto de librerías son las más utilizadas para nuestro caso, sin utilizar otras herramientas/librerías más complejas que el ejercicio prohíbe utilizar.

---

3. Hoja de Ruta del Desarrollo

3.1 Fase 1: Configuración Inicial

- Configuración del entorno de desarrollo (Python, OpenCV, NumPy)
- Conexión de la cámara mediante Camo
- Pruebas de captura de video en tiempo real

3.2 Fase 2: Detección de Cartas

- Implementación de segmentación por color HSV para detectar el tapete verde
- Desarrollo del algoritmo de detección de contornos
- Implementación de transformación de perspectiva para normalizar las cartas

3.3 Fase 3: Primer Intento de Reconocimiento (Template Matching)

- Creación de plantillas sintéticas para valores y palos
- Implementación de template matching con correlación normalizada
- **Resultado**: Precisión insuficiente, especialmente en la detección de palos

3.4 Fase 4: Segundo Intento (Cartas Completas + Múltiples Métodos)

- Captura de plantillas de cartas completas (52 cartas)
- Combinación de correlación, diferencia absoluta e histogramas
- **Resultado**: Mejora moderada, pero confusión entre cartas similares (7-8, etc.)

3.5 Fase 5: Solución Final (Diferencia Absoluta Simple)

- Adopción del método de diferencia absoluta de píxeles
- Umbralización binaria de imágenes antes de comparar
- Prueba automática de ambas orientaciones (0° y 180°)
- **Resultado**: Alta precisión en el reconocimiento

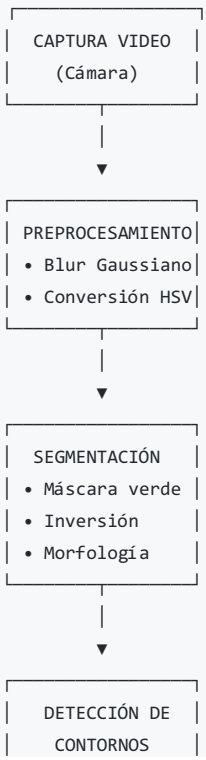
3.6 Fase 6: Optimización y Limpieza

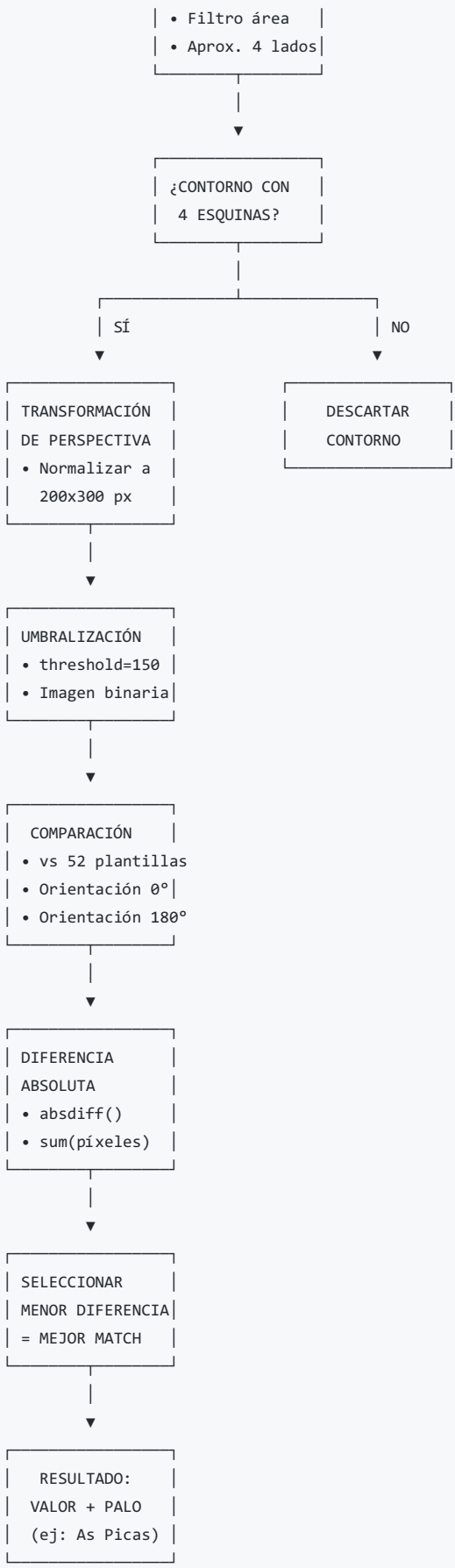
- Eliminación de código obsoleto
- Documentación del código
- Preparación de la memoria técnica

---

4. Solución

4.1 Diagrama de Decisión para Clasificación de Cartas





4.2 Secuencialización de Operaciones

4.2.1 Preprocesamiento de Imagen

```
 blurred = cv2.GaussianBlur(image, (5, 5), 0)
hsv = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)
```

Función	Parámetros	Justificación
`GaussianBlur`	kernel (5,5)	Tamaño moderado que reduce ruido sin perder detalles de bordes. Un kernel más pequeño dejaría ruido; uno más grande difuminaría los bordes de las cartas.
`cvtColor`	BGR→HSV	El espacio HSV separa el tono (H) del brillo (V), haciendo la segmentación por color más robusta ante cambios de iluminación.

4.2.2 Segmentación del Fondo Verde

```
GREEN_HSV_LOWER = (35, 40, 40)
GREEN_HSV_UPPER = (85, 255, 255)
green_mask = cv2.inRange(hsv, GREEN_HSV_LOWER, GREEN_HSV_UPPER)
card_mask = cv2.bitwise_not(green_mask)
```

Parámetro	Valor	Justificación
H mínimo	35	Límite inferior del verde en HSV (evita amarillos)
H máximo	85	Límite superior del verde (evita azules/cyan)
S mínimo	40	Requiere saturación mínima para evitar grises
V mínimo	40	Requiere brillo mínimo para evitar negros

4.2.3 Operaciones Morfológicas

```
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5, 5))
card_mask = cv2.morphologyEx(card_mask, cv2.MORPH_OPEN, kernel, iterations=2)
card_mask = cv2.morphologyEx(card_mask, cv2.MORPH_CLOSE, kernel, iterations=2)
```

Operación	Parámetros	Justificación
`MORPH_ELLIPSE`	5x5	Kernel elíptico para bordes suaves, mejor que rectangular para objetos redondeados
`MORPH_OPEN`	2 iteraciones	Elimina pequeños puntos blancos (ruido) sin afectar las cartas
`MORPH_CLOSE`	2 iteraciones	Rellena pequeños agujeros negros dentro de las cartas

4.2.4 Detección de Contornos

```
contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

Parámetro	Valor	Justificación
Modo	`RETR_EXTERNAL`	Solo contornos externos, ignorando agujeros internos (símbolos de las cartas)
Método	`CHAIN_APPROX_SIMPLE`	Comprime segmentos horizontales/verticales, reduciendo memoria

4.2.5 Aproximación Poligonal

```
perimeter = cv2.arcLength(contour, True)
epsilon = 0.02 * perimeter
approx = cv2.approxPolyDP(contour, epsilon, True)
```

Parámetro	Valor	Justificación

Parámetro	Valor	Justificación
epsilon	2% del perímetro	Balance óptimo: suficiente para aproximar a 4 lados, pero no tanto como para perder precisión en las esquinas

4.2.6 Transformación de Perspectiva

```
CARD_WIDTH = 200
CARD_HEIGHT = 300
dst_points = np.array([[0,0], [CARD_WIDTH-1,0],
                       [CARD_WIDTH-1,CARD_HEIGHT-1], [0,CARD_HEIGHT-1]])
M = cv2.getPerspectiveTransform(corners, dst_points)
warped = cv2.warpPerspective(image, M, (CARD_WIDTH, CARD_HEIGHT))
```

Parámetro	Valor	Justificación
Ancho destino	200 px	Suficiente resolución para distinguir detalles, sin exceso de procesamiento
Alto destino	300 px	Mantiene proporción aproximada de carta estándar (63x88mm ≈ 0.716)

4.2.7 Umbralización para Comparación

```
_, thresh = cv2.threshold(gray, 150, 255, cv2.THRESH_BINARY)
```

Parámetro	Valor	Justificación
Umbral	150	Valor medio-alto que separa bien el fondo blanco de la carta (>150) de los símbolos oscuros (<150)
Tipo	`THRESH_BINARY`	Convierte a imagen binaria pura (0 o 255), eliminando variaciones de gris

4.2.8 Comparación por Diferencia Absoluta

```
diff = cv2.absdiff(card_thresh, template_thresh)
score = np.sum(diff)
```

Operación	Justificación
`absdiff`	Calcula la diferencia absoluta píxel a píxel entre dos imágenes
`np.sum`	Suma total de diferencias. Menor suma = imágenes más similares

Ventajas de este método:

- Simplicidad: solo 2 líneas de código
- Robustez: funciona bien con imágenes binarizadas del mismo tamaño
- Velocidad: operaciones vectorizadas muy rápidas

---

5. Otras Tareas Realizadas

5.1 Herramienta de Captura de Plantillas

Desarrollé una herramienta interactiva (`capture_cards.py`) para capturar las 52 cartas de la baraja como plantillas de referencia. Esta herramienta:

- Detecta automáticamente la carta en el tapete verde
- Permite rotar la imagen 180° si está invertida
- Guarda la plantilla con nombre estandarizado (ej: `A_picas.png`)
- Muestra el progreso de captura (X/52 cartas)

5.2 Detección de Orientación

Implementé la detección automática de orientación probando la carta tanto en posición normal como rotada 180°, seleccionando la orientación con menor diferencia respecto a las plantillas.

5.3 Sistema de Calibración

El sistema incluye una función de calibración del color verde (tecla 'C' durante la ejecución) que permite ajustar los rangos HSV mediante trackbars para adaptarse a diferentes condiciones de iluminación.

5.4 Documentación del Código

Todo el código está documentado con docstrings explicando:

- Propósito de cada función
- Parámetros de entrada
- Valores de retorno
- Justificación de decisiones técnicas

---

6. Conclusiones

El sistema desarrollado cumple con los requisitos del reto:

1. **Reconocimiento básico**: Identifica correctamente valor y palo de cartas individuales
2. **Múltiples cartas**: Capaz de procesar varias cartas simultáneamente
3. **Sin IA/ML**: Utiliza exclusivamente técnicas clásicas de procesamiento de imágenes
4. **Tiempo real**: Funciona a ~30 FPS con reconocimiento instantáneo

La lección más importante del desarrollo fue que **las soluciones simples a menudo superan a las complejas** . El método final de diferencia absoluta de píxeles resultó más efectivo que los intentos anteriores con template matching multicapa y combinación de métricas.

---

7. Estructura del Proyecto

```
proyectoVision/
├── main.py              # Programa principal
├── card_detector.py     # Detección de cartas (segmentación, contornos)
├── card_recognizer_v3.py # Reconocimiento (diferencia absoluta)
├── config.py            # Configuración de parámetros
├── capture_cards.py     # Herramienta de captura de plantillas
├── requirements.txt     # Dependencias Python
├── README.md            # Documentación de uso
├── MEMORIA_TECNICA.md   # Este documento
└── templates/
    └── cards/           # 52 plantillas de cartas
```

---

8. Instrucciones de Uso

Instalación

```
pip install -r requirements.txt
```

Capturar plantillas (primera vez)

```
python capture_cards.py
```

Ejecutar reconocimiento

```
python main.py
```

Controles

- **ESC**: Salir
- **S**: Capturar imagen
- **D**: Mostrar/ocultar depuración
- **C**: Calibrar color verde